High-Performance Computing using Apache Ignite Hadoop Accelerator

Chaya Shivalingegowda, Srinivas Chevula, V. Y. Jaya Sree

Abstract: : Recently, IT has popularized the term high performance computing. In STEM High Performance Computing is used not only to model complicated calculations, but also to improve the applications, reduce costs and reduce development times, in order to solve large problems, in a much higher level of performance than a normal computer. In simple terms, the memory computing primarily relies on keeping data in the RAM as storage implies at high speeds. The HPC allows big data to be processed as quickly as possible.

Keywords : Big Data,High Performance , Apache Ignite, RAM

I. INTRODUCTION

High- powerful calculations are used not only to measure complex calculations, but also to boost application performance. HPC facilitates as fast as possible the storage of large data. By using several computers together, we can achieve 3-4 times the performance of a system in one cluster. Flash storage such as SSDs can work if we need a 5-6x improvement in performance.. But if we need more than 10 times the speed of the performance, we need to look for a different solution: computer in memory. The high-efficiency cloud application provides support for simplified data parallel applications [7], physical applications [5]. Several studies and research are being improved [6]. Apache Ignite provides a simple interface for programmers to work in real-time for large-scale data sets and other facets of storage computing. Ignite provides Information Center, Virtual Cloud, Network Portal, Grid Sharing, and Big Data Accelerator. By adding additional nodes, a Ignite Cluster setup may increase horizontally. The RDBMS Ignite Grid is able to persist in cache, even in NoSQL cache.

Ignite promotes Cache as an organization-wide service that provides multiple applications with access to maintained in-memory cache rather than a disk dependent server. High efficiency Apache Hadoop accelerator. In order to improve the performance of all big data analyses projects, the Apache Hadoop job tracker and IGFS can be replaced by Ignite. Distributed computing: Ignite offers easy mechanisms for moving computational and information for higher performance across the network. Streaming: Ignite allows to use continuous streams of information.

Revised Manuscript Received on December 13, 2019.

***** Correspondence Author

Chaya Shivalingegowda, Asst. Professor, Dept. of ECE, Kalsekar Engineering College, New Panvel, Mumbai, India.

Srinivas Chevula, Dept. of Electromics and Telecommunication engineering, , AIKTC, Mumbai, India

Prof. P V Y Jayasree, Dept. of EECE, GIT, GITAM, Visakhapatnam, A.P, India.

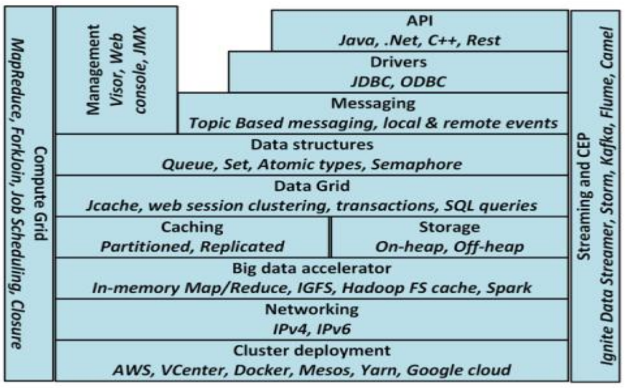
Compute Grid
MapReduce, ForkJoin, Job Scheduling, Closure
Management
Visor, Web console, JMX
API
Java, .Net, C++, Rest
Drivers
JDBC, ODBC
Messaging
Topic Based messaging, local & remote events
Data structures
Queue, Set, Atomic types, Semaphore
Data Grid
Jcache, web session clustering, transactions, SQL queries
Caching
Partitioned, Replicated
Storage
On-heap, Off-heap
Big data accelerator
In-memory Map/Reduce, IGFS, Hadoop FS cache, Spark
Networking
IPv4, IPv6
Cluster deployment
AWS, VCenter, Docker, Mesos, Yarn, Google cloud
Streaming and CEP
Ignite Data Streamer, Storm, Kafka, Flume, Camel

Fig .1 Architecture overview of Ignite

The Ignite architecture has advanced features that can be used in a large number of different architectural patterns and styles. We can see Ignite as a collection of in-memory components to improve the performance and scalability of the application. The Fig.1 schematic represents the basic functionalities of Ignite.

Ignite design implies that the entire system itself is both inherently available and massively scalable. Its internode communication allows all nodes to receive updates without the need for a master coordinator quickly. Systems can be modified to increase the amount of *RAM*.

With this approach, it can perform on the same JVM with the application. Ignite server run along with the application in the same JVM and joins with other nodes of the grid. If the application shuts down, its node can operate on the same JVM as the request with this form. With the application the Ignite server runs in the same JVM and adds to the other grid nodes. Ignite server also shuts off if the application is shut down. Client nodes operate in another JVM, and user nodes are linked to servers on a remote basis. It provides three different approaches to caching topology: Partitioned, Replicated and Local. The default cache topology is partitioned, without any backup option. In partitioned caching topology Ignite cluster partitions the cached data to distribute the load across the cluster. By partitioning the data evenly, the size of the cache and the processing power grows linearly with the size of the cluster. The responsibility for managing the data is automatically shared across the cluster. Every node or server in the cluster contains its primary data with a backup copy if defined. In the replicated caching topology, the goal here is to get extreme performance. This technique replicates cache information for all cluster participants. The information can be used without any wait since it is distributed to every cluster node. It guarantees optimum read-access speed; each participant may access the data from his or her own memory.

The downside is that frequently written documents are very costly. Using the new version to update a distributed buffer. With this approach, cache data is *replicated* to all members of the cluster. Since the data is replicated to each cluster node, it is available for use without any waiting. This provides highest possible speed for read-access; each member accesses the data from its own memory. The disadvantage is that frequent writes are very expensive. Updating a replicated cache requires using the new version to all other cluster members.

A. Distributed Key-Value Store

Ignite gives different API and can be used as a pure in memory store. Different nodes in the cluster system store different parts of the indexes and data. Ignite provides different APIs and can be used in memory storage for simple purposes. Also, Ignite can store data in memory as well as on the disk, and thus more data can be stored than using just the secondary storages.

B. Co-locate Processing

The traditional systems use the client server way, where it fetches data from the server to client and here it gets processed. This way moving data from network is very expensive operation. A better way is to co-locate which get the processing to the servers where the computations take place. Thus avoiding expensive network round trips. The traditional systems use the client server way, where data is collected from the server to the client and processed here. This is a very difficult way to move information from a network. A better way is to co-locate the servers where calculations are carried out with processing. Thus, costly network trips are avoided.

C. ACID Transactions

Ignite supports fully distributed ACID transactions. It applies to both the in memory store as well as the secondary storages. Transaction in Ignite can span multiple cluster nodes, caches (aka. tables) and partitions. The purpose of this to propagate updates to the disk in the append-only mode, which is the fastest way to persist data to disk. It provides a recovery mechanism for failure scenarios when a single node or the whole cluster go down. A system can be recalled to the latest successfully transactions.

D. Machine Learning

Ignite Machine Learning (ML) tools allow using different models without the need for costly data handling and transfers. Ignite ML depends on durable memory that gets huge scalability for machine learning computations and removes the time required for Extract, Load and transform. It allows machine learning training and concludes directly on data stored across the systems. Next, Ignite provides a host of Machine learning procedures that can be used for Ignite's collocated distributed processing. These use-cases provide in memory speeds and massive storage space for incoming data, without requiring the data to be moved into another store. Thus this can be used for real time machine learning analytics.

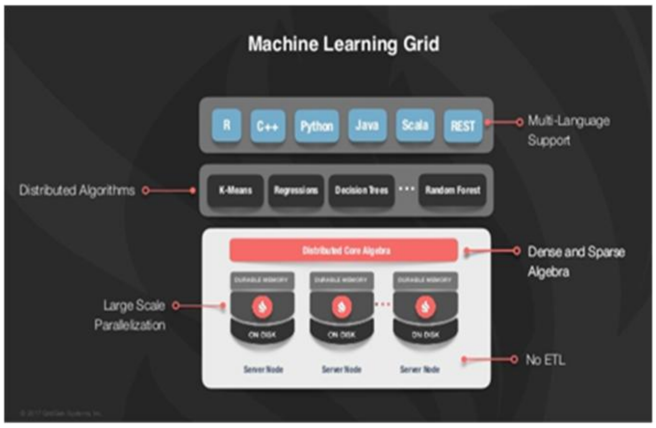

Fig 2. Real Time Machine learning Analytics

E. Data Loading and Streaming

Ignite data loading and streaming capabilities allow ingesting large finite as well as never-ending volumes of data in a scalable and fault-tolerant way into the cluster. The rate at which data can be injected into Ignite is very high.
Ignite data loading and streaming technologies enable massive, finite and never-ending amounts of information to be absorbed into the cluster in an elastic and defect resistant manner. In a moderately large cluster, the rate at which information is injected in the ignite is very high and easily exceeds millions of events per second. Server systems use Ignite Data to move flows (finite or continuous) of information to Ignite. In addition, a moderately large cluster can easily overcome millions of events per second. Using Ignite Data Streamers[4], client systems are transferring the streams (finite or continuous) of data into Ignite. Data is automatically partitioned into nodes, and the same amount of data is allocated to each node. Streaming data can be processed in a co-located manner directly on the Ignite data nodes.

F. System Parameters

Intel Xeon (R) CPU E5 v4 @ 2.20GHThread(s) per core: 1Core(s) per socket: 22Socket(s): 2 RAM size: 62GB Secondary storage: 1TB SSD Cache layers: L1, L2, L3

G. Different types of benchmark tests performed:

There exist several kind of benchmark test to test the performance of high performance computing. Few of them are explain in text briefly:

II. RESULTS

We perform several simulation using apache ignite to test the performance. When running the benchmark for just 1 client thread and 10 jobs, the CPU has utilized around 2%, and the average number of operations were around 1559/sec when the numbers of client threads were increased to 256, the CPU utilization was around 38% and throughput was around 90549 op/sec.

1. **PutGetBenchmark**: Benchmarks atomic distributed cache put and get operations together
2. **PutGetTxBenchmark**: Benchmarks transactional distributed cache put and get operations together
3. **SqlQueryBenchmark**: Benchmarks distributed SQL query over cached data.
4. **SqlQueryJoinBenchmark**: Benchmarks distributed SQL query with a Join over cached data
5. **Execute Benchmark: B**enchmarks execute operations
6. **BroadcastBenchmark**: Benchmarks broadcast operations
7. **SqlQueryPutBenchmark**: Benchmarks distributed SQL query with simultaneous cache updates

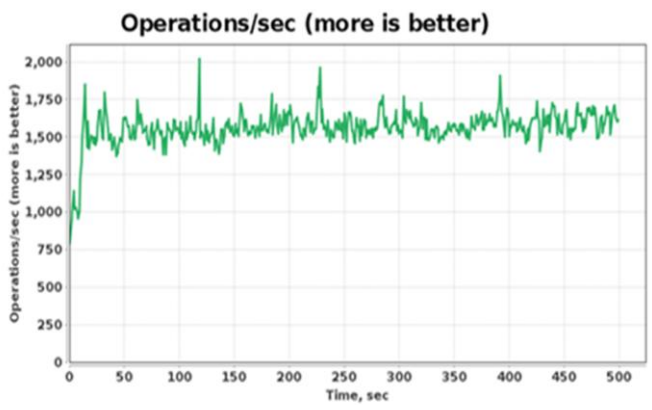

Fig 3. CPU Utilization around 2%

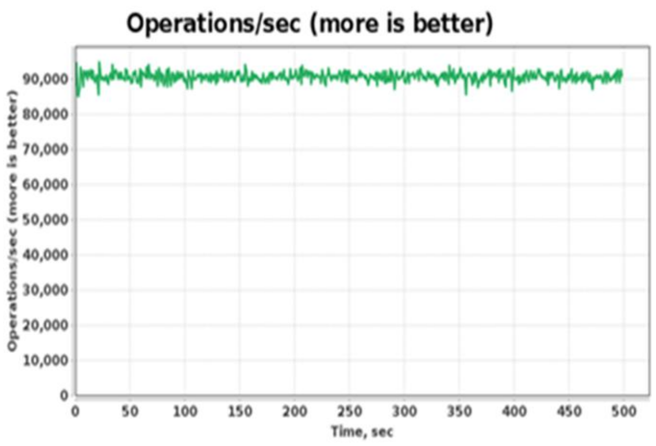

Fig 4. CPU Utilization around 38%

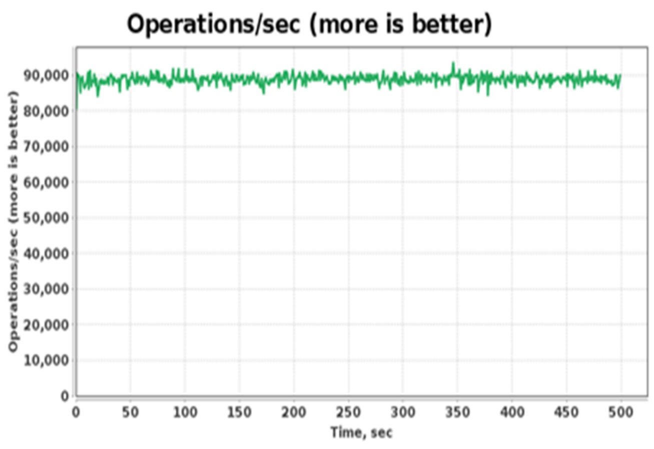

Fig 5. CPU usage paging and Memory Usage

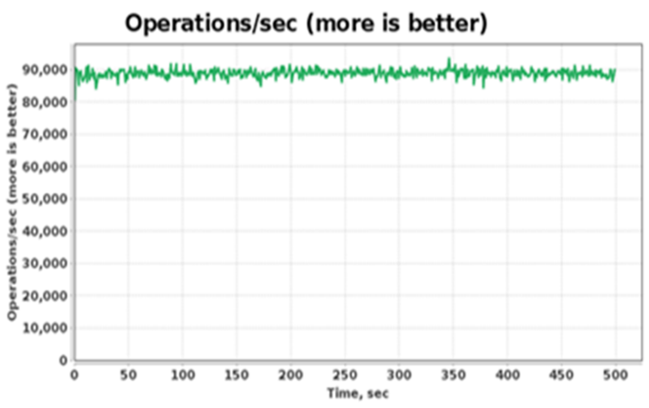

Fig 6. CPU usage paging and Memory Usage

III. OPTIMAL PERFORMANCE CACHEPUTGET BENCHMARK

In almost all cases, 1 or 2 cores were getting utilized ~95% (60 user + 35 sys), rest all cores (42) were idle at around 75%.Cache Mode: Partitioned, Atomicity Mode:Atomic,Page Size : 4KB Execution time: 300 secs

IV. INTEGRATION OF APACHE IGNITE WITH APACHE KAFKA STREAMER

Apache Kafka is an open source distributed framework and can be used for building real time data pipeline and streaming apps. To analyze the number of messages that can be streamed into a partitioned Ignite Cache use configurations that yields maximum throughput. Apache Kafka is a distributed open source frame that can be used to build real-time data pipelines and stream applications. In order to analyze the number of messages to be streamed in a partitioned Ignite Cache, use configurations that provide maximum performance. Apache Ignite Kafka Streamer supports synchronization from the cache Kafka to the server Ignite. Both the two approaches used in this streaming can be used:
• use Ignite sink to link Kafka Connected functionality;
• importing the Kafka Streamer unit into the Maven project and instanting KafkaStreamer for data streaming

Stream data with Kafka connect
Ignite Sink Connector helps to put data from Kafka to Ignite and the procedure is as following:

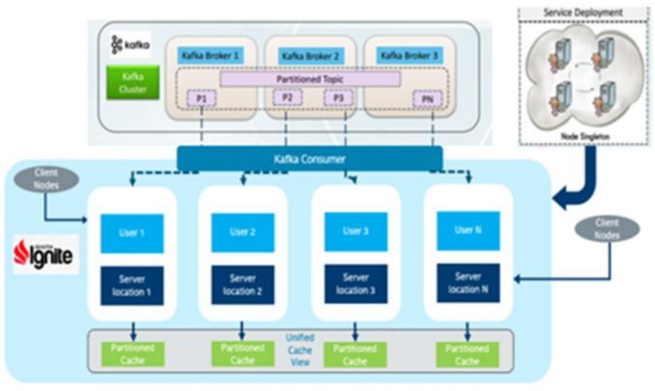

Fig 7. Kafka Streamer for data streaming

V. SETTING UP AND RUNNING

1. Put the jar files on Kafka's classpath

```
ignite-kafka-x.x.x.jar <-- with IgniteSinkConnector
ignite-core-x.x.x.jar
cache-api-1.0.0.jar
ignite-spring-1.5.0-SNAPSHOT.jar
spring-aop-4.1.0.RELEASE.jar
spring-beans-4.1.0.RELEASE.jar
spring-context-4.1.0.RELEASE.jar
spring-core-4.1.0.RELEASE.jar
spring-expression-4.1.0.RELEASE.jar
commons-logging-1.1.1.jar
```

2. Prepare worker configurations

```
bootstrap.servers=localhost:9092

key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.storage.StringConverter
key.converter.schemas.enable=false
value.converter.schemas.enable=false

internal.key.converter=org.apache.kafka.connect.storage.StringConverter
internal.value.converter=org.apache.kafka.connect.storage.StringConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
```

3. Prepare connector configurations

```
# connector
name=my-ignite-connector
connector.class=org.apache.ignite.stream.kafka.connect.IgniteSinkConnector
tasks.max=2
topics=someTopic1,someTopic2

# cache
cacheName=myCache
cacheAllowOverwrite=true
igniteCfg=/some-path/ignite.xml
singleTupleExtractorCls=my.company.MyExtractor
```

4. Start Zookeeper server

```
bin/zookeeper-server-start.sh config/zookeeper.properties
```

5. Start Kafka server

```
bin/kafka-server-start.sh config/server.properties
```

6. Provide some data input to the Kafka server

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic test --property parse.key=true --property key.separator=,
k1,v1
```

7. Start connector, in a standalone mode as follows

```
bin/connect-standalone.sh myconfig/connect-
standalone.properties myconfig/ignite-connector.properties
```

8. Check the value is in the cache. For example, via REST API

```
http://node1:8080/ignite?cmd=size&cacheName=cache1
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.apache.ignite</groupId>
            <artifactId>ignite-kafka</artifactId>
            <version>${ignite.version}</version>
        </dependency>
        ...
    </dependencies>
    ...
</project>
```

APACHE IGNITE HADOOP ACCELERATOR

Apache Ignite Hadoop Accelerator provides a set of modules that enable Hadoop job execution and the running of your File System. Ignite In-Memory MapReduce enables the storage of information that is processed in a Hadoop File System to be easily paralleled. It eliminates the overhead associated with job-trackers, task-trackers and low-latency, HPC-style distributes. First, Ignite delivers a range of machine learning techniques which can be used for the decentralized storage of Ignite. Such case stores have large storage room to accept information at memory levels without the need to transfer the data to a different store. It can therefore be used for deep learning research in real time.

VI. CONCLUSION

We have benchmarked Apache Ignite and analyzed scenarios where Ignite results in optimized performance, which can eventually be used in different use-cases, a couple of them being with Apache Kafka live data streamers and Apache Hadoop file systems and MapReduce jobs. We have also studied some constraints holding for our scheme. ACID transactions in pure in-memory distributed state being one of them.

Ignite proves to be very efficient as compared to other in-memory platforms and improves streaming/distributed data of clusters' access considerably.

Purely in-memory asynchronous ACID transfers are one of them. Ignite is highly effective than other in-memory systems and greatly facilitates the streaming / distribution of cluster data.

REFERENCES

1. Santos, Adrián, Francisco Almeida, and Vicente Blanco. "Lightweight web services for high performace computing." European Conference on Software Architecture. Springer, Berlin, Heidelberg, 2007..
2. Distributed Persistence Ignite https://ignite.apache.org/arch/persistence.html
3. In-Memory Database Ignite https://ignite.apache.org/use-cases/database/in-memory-database.html
4. Data Streamers Stream large amounts of data into Ignite caches.https://apacheignite.readme.io/docs/data-streamers
5. Zavala-Aké, J. Miguel. "A high-performace computing tool for partitioned multi-physics applications." (2018).
6. LI, Cheng-hua, et al. "MapReduce: A new programming model for distributed parallel computing [J]." Computer Engineering & Science 3 (2011): 026.

7. Bingxiang, Gui, and Feng Hongcai. "An New Data Parallelism Approach with High Performace Clouds." 2009 Pacific-Asia Conference on Circuits, Communications and Systems. IEEE, 2009

AUTHORS PROFILE

Chaya.S, Now Ph. D Research scholar (part time) at the Electronics & Communication Department of GITAM Vizag. She has a Assisitant professor at the New Panvel, Mumbai, Kalsekar Engineering College.

Mr.Srinivas Chevula is student of AIKTC in Electronics & Telecommunication completing his degree in 2020 from Mumbai University.His Area of Interset is DataScience and MAchine Learning.

P.V.Y Jayasree received the B.E degree in Electronics and Communication Engineering from GITAM University, Andhra Pradesh, India, in 1989, Master Degree in Electronic Engineering from Andhra University, Vizag, AP, India in 1999, and Ph.D degree in EMI EMC from JNTU Kakinada, India in 2010. She is currently Professor, Head of Department of Electrical Electronics and Communication Engineering with GITAM Univeristy, Visakhapatnam, AP, India. She has author or co-author over 32 Journal Papers and 35 conference papers. She has co-authored 7 books. Her current research interests includes antenna design for 5G applications, tunnel FET, microwave passive circuits and components, low power SRAM cell design, and phased array antennas. Dr. P.V.Y. Jayasree has been the Chair of Board of Studies with GITAM University since 2018.